A Formal Method Based Re-Implementation Concept for PLC Programs and Its Application

Mohammed Bani Younis and Georg Frey
University of Kaiserslautern
Erwin-Schrödinger-Str. 12
67663 Kaiserslautern, Germany
{baniy,frey}@eit.uni-kl.de

Abstract

Programmable Logic Controllers (PLCs) are still the workhorse of industrial automation. The programming and adaptation of the software for PLCs – i.e. the implementation of the control algorithms – are expensive and cumbersome tasks. Today the re-implementation of an existing PLC program on a new platform, as it is supported by industrial tools on a low level, requires considerable manual re-working by a specialist. Known re-implementation processes do not allow a fully automatic transfer to a new platform. Furthermore they lack re-documentation of the code which makes it hard to understand and to follow the implemented algorithms. To avoid these problems, this paper proposes a re-implementation concept for existing PLC programs based on formal methods. An application of this concept to a lab system is presented to assess its applicability.

1. Introduction

Programmable Logic Controllers (PLCs) have initiated the success of automation in manufacturing systems. Due to their flexible programming frequent changes in the automation system are feasible. The need for changes arises clearly to meet new production demands which can not be fulfilled by the PLC programs already running on the machine. The need to re-implment already existing PLC programs into a new platform is a new important issue related to PLCs. One reason is that the above mentioned extensions at some point may lead to programs that need a more powerful controller to run on. Another important reason for the re-implantation is given when the PLC hardware is no longer produced or supported by the vendor. These obstacles have inspired researchers to investigate ways to analyze, verify, and simulate PLC programs on an abstract level before migrating them to the controller running on the physical plant [1, 2, 3, 4]. The design for re-usability using modular modeling techniques is another important direction of research in this area.

This work is dedicated to delineate the re-implementation of a formal description of STEP 5 (also known as S5) PLC programs into an IEC 61131-3 environment. Figure 1 depicts the re-engineering concept super-imposed on the taxonomy of [5]. The depicted programming tools are only used to export the PLC program from the source system and to load it down to the target system.

Figure 1: Re-Engineering Concept.

The presented re-implementation method completes the re-engineering process that starts with the reverse engineering of the S5 code into a formal description. Note that the reverse engineering starts from a given PLC code which is converted into XML (raw XML and XML with instruction identification [6]). Thus the re-implementation is the last step in the re-engineering which in this scope will play the role as the essence of this process.

Though the conversion of a given S5 PLC programs can be done on the low level using industrial tools [7, 8]. This re-use is unwieldy and the outcome concerning its efficiency is imprecise. Furthermore this conversion is rigid concerning the mapping from source to target PLC platform (e.g. Siemens S5 to Siemens S7 in [7]). The presented approach allows besides the re-implementation also the verification, analysis and etc. Furthermore, due to the intermediate formal description, it is flexible concerning the mapping of the source to the target PLC.
The presented re-implementation concept utilizes the reverse engineering steps presented in [6, 9, 10]. The re-engineering consists of the following main steps which will be detailed in the following sections:

1. Reverse-engineering of the given PLC program through formal methods using Internet technologies.
2. Evolution of the PLC structure of S5 to IEC 61131-3. This step provides a generic basis for all PLC programs to be transferred from S5 to IEC 61131-3.
3. Mapping of a given S5 PLC program structure with its blocks into IEC 61131-3. After this step the single PLC blocks of S5 are converted to new blocks compliant to the structure of IEC 61131-3.
4. Transferring the dynamic semantics of the different blocks. The code of the blocks together with the variable declarations is mapped to the new blocks and variables based on formal models (finite state machines, FSM).

The rest of the paper is structured as follows. Section 2 gives a recall of the reverse-engineering step. Since the second and third step mentioned above are closely related to each other, the process for evolving will be explained together with structure mapping in Section 3. This gives the basis for the conversion concept in Section 4. A case study on a lab plant is given to evaluate this process in Section 5. Section 6 concludes this paper and gives an outlook on future work.

2. Reverse-Engineering of S5 PLC programs

2.1. Formalization Concept

This step allows the abstraction of the PLC code written in Instruction List (IL) into Finite State Machines (FSM). It is an extension to previous work described in [6]. It is done in Java and it can be illustrated by the following steps (cf. Figure 2):

**Step 1:** The PLC code in IL is converted to raw XML [6, 10, 11]. This raw XML is then converted to a core XML using XSL. The core XML contains in addition to the attributes specified in the IL syntax (Address, Label, Instruction, and Operand) other attributes generated for the reverse-engineering (InstructionId, Type, Condition, and Denotation). The type of the instruction is the crucial attribute in the next step.

**Step 2:** Splitting the type attribute in the XML to obtain a description in the form of IF-THEN-ELSE statements. The type splitting in the XML becomes very essential for constructing the algorithms. The conversion into IF-THEN-ELSE is performed according to the algorithms given in [9, 11].

**Step 3:** Constructing IF-THEN-ELSE statements with using the Document Object Model (DOM) [12] which makes it easy to extract information from XML.

**Step 4:** Developing the FSM in XML format from the IF-THEN-ELSE statements. The resulting FSM completely describes the given program in a formal way.

The IF-THEN-ELSE statements due to the abstraction of the PLC algorithm are as follows (cf. Figure 4):

```
IF E38.1 AND E38.2 OR E38.1 AND E38.3 OR E38.2 AND E38.3
THEN M100.0 = 1
ELSE M100.0 = 0
```

**Figure 3: Binary algorithm.**

**Figure 4: IF-THEN-ELSE description for the example.**

These IF-THEN-ELSE statements can be optimized to give M100.0 = E38.1 AND E38.2 OR E38.1 AND E38.3 OR E38.2 AND E38.3 which is again transformed into an XML describing the FSM of the PLC algorithm as shown in Figure 5. Using Graphviz [14], the FSM can be visualized as given in Figure 6.
The main idea of the evolution and structure mapping of STEP 5 to IEC 61131-3 is illustrated as in the following:

- **OB1**: the cyclic processing of the PLC programs starts from this block which resembles the functional task of the Program in the new PLC. Therefore, the Program POU incurs the operational semantics of the OB1.
- **Other OBs**: these OBs are assigned to perform some tasks in STEP 5. They can be converted to programs or function blocks depending on their operational semantics. As an example OB20, OB21, OB22 are used for initialization and new start. These blocks are used only once and are not part of the cyclic behavior of the PLC. Details for their conversion will be shown later.
- **PB and FB**: these blocks are converted to FBs and hold the same name as it was assigned in S5. The difference between FBs and PBs is that, the FBs are function blocks that can be frequently instantiated. This generates no problem in case of S5 when called with the same name. But in the case of IEC 61131-3 the output values will be overwritten whenever called with the same name. This is why an index is added to the declaration of the function block each time it is instantiated.
- **Data Blocks**: the DBs are converted into an Array in IEC 61131-3 declared as a global variable in the Main Program [17]. The values of the DB will be taken into the array and are influenced when accessed by other function blocks. Data used in PBs and FBs of S5 have to be declared as external variables in the corresponding function block in IEC 61131-3.
- **Symbol Table**: the Symbol Table in STEP 5 undertakes the task of the global variable declarations. It is in a tabular form consisting of three columns, containing the Operand, Symbol, and Description which is used to comment on the variable or its symbol. In STEP 5 it is possible in a given code of a block through its programming tool to switch from the addressed operands to its symbols. Table 1 exposes a shortened view of a symbol table of a PLC program. Note that the E, A, M stands for the Eingang (input), Ausgang (output) and Merker (internal variables) respectively. The numbers following these letters expresses the physical address of the operands. These variables have to be specified in the main program of the IEC 61131-3 project. Furthermore, they have to be declared as external variables in case used in the corresponding function block.
- **Other elements in S5 which can not be mapped to a direct counterpart in the IEC 61131-3 have to be re-programmed as functions or function blocks and to be integrated into the project library. This is the case also for the non-binary instructions of S5.
### 4. Conversion Concept of the Dynamic Semantic

From Figure 7 the conversion process commence from the FSM in XML format of the PLC modules where the XML describes the FSM gained through the conversion of PLC algorithms. As elucidated in the figure this process is recursive to cover all the FSMs of the modules contained in the PLC program fed to the conversion process. Using DOM [12] the information constituted in the XML is extracted and saved in a new document. This step is followed by parsing this XML tree to search for the input and action tags where S5 keywords of the XML are of importance to verify the kind of conversion to be done. The input for the conversion into IEC code is the tree structure along with the S5 to IEC keywords mapping (cf. Table 2). The symbol table is needed to substitute the physical declaration of the variables into its symbol. The symbol table and the keywords mapping are described in the implementation as a hashtable which maps keys to values. Data blocks are integrated in the conversion process in case they exist. The output of the conversion method is the elements of the IEC 61131-3 program (such as the Program along with its global variables of inputs, outputs, and internal variables).

As mentioned before a program will be generated by this conversion which actually takes over the task of the OB1. This process toward the conversion will be repeated until all the PBs and FBs are encompassed. Another important aspect to encounter is the initialization through OBs indexed 21, 20 and 22, since they have no direct counterpart in IEC. To inherit them in the conversion process a global variable called Init_PLC assigned to true in its initial declaration is used. This variable is set to false in the corresponding function block which guarantees the single execution of it in the cyclic process. After this conversion is complete the new POUs are saved in a project directory or a text of the complete set of POUs is generated and saved with an exp extension (the export project format of [8]). Illustrating examples will be given throughout the next sections for binary algorithms, timers and counters, and non-binary algorithms.

### Table 1: Symbol Table.

<table>
<thead>
<tr>
<th>Operand</th>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E 0.0</td>
<td>1K1</td>
<td>STEUERUNG AUS/EIN</td>
</tr>
<tr>
<td>E 0.2</td>
<td>1S10</td>
<td>LAMPE TEST</td>
</tr>
<tr>
<td>A 0.0</td>
<td>1H5</td>
<td>STEUERUNG</td>
</tr>
<tr>
<td>A 0.2</td>
<td>1H7</td>
<td>EINRICHTEN</td>
</tr>
<tr>
<td>M 0.0</td>
<td>M0.0</td>
<td>VKE = 0 FUER BCD WANDLUNG + VORZEICHEN</td>
</tr>
<tr>
<td>M 0.2</td>
<td>M0.2</td>
<td>RESET STORRMELDUNGEN</td>
</tr>
<tr>
<td>MB 120</td>
<td>MB120</td>
<td>STORRUNGEN'S WALZE FUER TEXTANZEIGE</td>
</tr>
</tbody>
</table>

### Table 2: S5 to IEC Keywords Mapping.

<table>
<thead>
<tr>
<th>S5 Keywords in FSM</th>
<th>IEC 61131-3</th>
<th>S5 Keywords in FSM</th>
<th>IEC 61131-3</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>AND</td>
<td>&lt;, &lt;=</td>
<td>LT, LE</td>
</tr>
<tr>
<td>ANDN</td>
<td>ANDN</td>
<td>&gt;, &gt;=</td>
<td>GT, GE</td>
</tr>
<tr>
<td>OR</td>
<td>OR</td>
<td>&lt;=</td>
<td>NE</td>
</tr>
<tr>
<td>ORN</td>
<td>ORN</td>
<td>=</td>
<td>:=, ST</td>
</tr>
<tr>
<td>+</td>
<td>ADD</td>
<td>-</td>
<td>SUB</td>
</tr>
<tr>
<td>*</td>
<td>MUL</td>
<td>/</td>
<td>DIV</td>
</tr>
<tr>
<td>Jump to</td>
<td>JMP</td>
<td>Call to</td>
<td>Call</td>
</tr>
</tbody>
</table>

### Figure 7: Conversion concept.

4.1. Conversion of Binary Algorithms

The conversion of binary algorithms occurs automatically from the FSM of the PLC module in XML description. This guarantees the direct conversion of the logic and dynamic of the FSM into IEC 61131-3. The operands are declared as Global Variables in IEC and in case no symbol assigned for this variable in the symbol table exists. If the variable is assigned to a matching symbol it will be declared through its symbol. The conversion of the semantic will occur by inserting the matching IEC instruction of the FSM keywords. An important issue in this conversion is nesting of logical instructions in case the output language for the conversion is IL. To solve this True and False after a
parenthesis have to be added since IEC does not allow an empty operand after a binary instruction. True will be added to a nested OR instruction of ANDs whereas False will be assigned to an AND nesting of ORs. An alternative to this conversion to avoid this issue is the direct conversion of the IF-THEN-ELSE into ST of the IEC. An illustrative example expressing the conversion of a binary algorithm from its FSM is depicted in Figure 8 (this XML is taken from Figure 5 which is the conversion of the code in Figure 3).

4.2. Conversion of Timers and Counters

The conversion of timers and counters does not start from the XML of the FSM. The transformation takes place by matching the timer and counter type to its counterpart in IEC following Table 3. The XML of the FSM is useful for the extraction of the condition which is relevant for the initiation of timer and counter. Other types of timers and counters have to be reprogrammed in IEC 61131-3 as function blocks (i.e. SI). Figure 9 shows an illustrating example of an impulse timer. An important issue in this transformation is the information about the condition, timer type and time value. The FSM helps to conceive the conversion process by visualizing the semantic of the timer type.

4.3. Conversion of Non-Binary and Data Blocks

The conversion of non-binary instructions occurs through the designated FSM in the XML format. The semantic of the non-binary instructions is migrated into a newly defined function block structured into IEC 61131-3 which will thereafter be connected to the project directory. This step allows the compatibility of the conversion to the input PLC. The variables of the FSM such as AKKU 1, AKKU2, OV and etc. are declared as global variables in the main program and as external variables in the corresponding function block. This newly generated function block will be invoked by the function block when it appears in the FSM.

An important component while working with non-binary function blocks (FBs) of S5 are the data blocks (DBs). These DBs are initiated by there invocation by the PBs which access the data variables inside it. After this access the PB calls another FB using these data variables as parameters.

Figure 10 illustrates an example for the conversion of both non-binary and data blocks of the code shown below. In this example the PB1 calls the DB2 and the FB2 (named as ADD). The arguments of the FB are DW3, DW4 and DW5 which are data elements of the DB. These data variables substitute the variables of the FB2. Note that the +F non-binary instruction is converted into a new function block called ADDS5 which reflects its semantic in S5. The data block which is supposed to be used for the first time in this PB is declared through an ARRAY in the program main. The data of the DB will be changed by its next use.
5. Case Study

5.1. Plant Description

The plant used to test the re-implementation and conversion application was a didactic Modular Production System (MPS) from FESTO [18]. The principal objective of the didactic prototype is to examine cylindrical work pieces for proper thickness and material type, drill a proper hole on each work piece and then sort them according to their material types. As the name implies, the plant consists of different modules. The modules are again grouped into four stations which are the Distribution, Testing, Processing and Storing (cf. schematic in Figure 11). A brief description of the four stations is given in [19].

During normal operation a work piece will be pushed out of the magazine by the feeder and transported to the test spot by the transport module. The lift module brings the work piece to the measuring module, and if the piece exhibits a tolerable thickness it is pushed by the cylinder mounted on the module to the conveyor, otherwise the lift moves down and pushes it to the slider which guides it to the scrap area. The conveyor guides a proper piece to the position 1 of the rotary table. The table then rotates till the piece comes under the drill module. After the drilling the rotary table moves further till it reaches the drilled hole checking module. After the piece is checked for correct drilling the table moves the processed piece to position 4 from where the crane can pick it up and move it to the proper repository. Apart from the normal operation, some other operating modes of the plant are implemented, namely, Initializing, Clean-up, Emergency stop, and Deadhead mode.

5.2. Conversion Process

The system handles 41 binary input and 32 binary output signals. It was originally controlled by a Siemens PLC programmed in STEP 5 IL. The control software to be formalized contains 21 blocks adding up to a total of about 1558 lines of IL code. These blocks contain binary, timer, counters and non-binary instructions. The PLC program was converted automatically using JAVA programs implementing all the steps described above to XMI which reflects the structural view of the program in UML (cf. Figure 12). A set of CFSMs (Communicating Finite State Machines) for the existing blocks were also attained through this conversion. An example of this conversion is of the OB1 of the S5 program from its code shown in Figure 13 into a CFSM visualized in XML is depicted in Figure 14.

The visualization of the FSM (cf. Figure 15) is done using SVG. Other blocks can also be visualized in the same way but for the re-implementation the FSM in XML format is of relevance.

Figure 10: Non-Binary and Data Block Conversion.

Figure 11: Schematic of the didactic plant.
The application developed in course of this work has given the ability to visualize the CFSMs through SVG [20] and [21]. Through this conversion a more comprehensive grasp on the controller running on the plant was achieved. It has also provided an important basis for the successful re-implementation of the existing control on a new controller running a programming system compliant with IEC 61131-3. The conversion into IEC 61131-3 was performed based on the structure of the given PLC into a set of XML format of the FSM. The FSM of OB1 is converted into the program of IEC 61131-3 as shown in Figure 16. The declaration of the global variables is not complete and it declares the variables according to their names in S5 since during the conversion the symbol table was not given to the input of the converter.
A project directory was generated as an output of the conversion and was tested on the FESTO machine using an Industrial PC (IPC) connected to another real time IPC (SMP 16 from siemens) [22]. With the help of this HW platform the IEC 61131-3 was able to drive the machine using a SoPlC (OpenPCS of infoteam software AG [23]). Later the behaviors of both of the PLCs running on the FESTO machine were compared empirically. The result of this comparison was that, the behavior of the PLC in S5 resembles that of the IEC 61131-3. This similarity exists though the cyclic behavior, the structure, and the processing way of these PLCs are different. Plugging S5 PLC to IEC and back on the controlled machine the controlled system behaves exactly in the same way as when resetting the currently running PLC. To conclude the migration of S5 PLC program into IEC 61131-3 was totally complete.

![Program Listing]

**Figure 16:** IEC 61131-3 Program converted from FSM of OB1.

### 6. Conclusions and Outlook

The paper presents an approach for the re-implementation of existing PLC programs based on formalization and visualization. The FSM in XML format is transformed into IEC 61131-3 POUs and a project is built to control the machine using the new controller. The re-implementation to a new platform is important to meet new manufacturing demands or to substitute a PLC whose HW is no longer produced. This re-implementation concept was tested on a didactic plant which was the basis for the visualization and migration of the STEP 5 PLC code running on the plant to a IEC 61131-3 compliant controller.

The next steps in the presented work are extension of the re-implementation step to further environments, inclusion of further source PLC types in the reverse-engineering step, and – most important – application of the method in an industrial setting.

### References


[8] 3S CoDesys home-page: http://www.3s-software.com/


[12] DOM web site http://www.w3schools.com/dom/


